

# HIBERNATE BATCH PROCESSING

[http://www.tutorialspoint.com/hibernate/hibernate\\_batch\\_processing.htm](http://www.tutorialspoint.com/hibernate/hibernate_batch_processing.htm)

Copyright © tutorialspoint.com

Consider a situation when you need to upload a large number of records into your database using Hibernate. Following is the code snippet to achieve this using Hibernate:

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Employee employee = new Employee(.....);
    session.save(employee);
}
tx.commit();
session.close();
```

Because by default, Hibernate will cache all the persisted objects in the session-level cache and ultimately your application would fall over with an **OutOfMemoryException** somewhere around the 50,000th row. You can resolve this problem if you are using **batch processing** with Hibernate.

To use the batch processing feature, first set **hibernate.jdbc.batch\_size** as batch size to a number either at 20 or 50 depending on object size. This will tell the hibernate container that every X rows to be inserted as batch. To implement this in your code we would need to do little modification as follows:

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Employee employee = new Employee(.....);
    session.save(employee);
    if ( i % 50 == 0 ) { // Same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

Above code will work fine for the INSERT operation, but if you are willing to make UPDATE operation then you can achieve using the following code:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

ScrollableResults employeeCursor = session.createQuery("FROM EMPLOYEE")
                                         .scroll();

int count = 0;

while ( employeeCursor.next() ) {
    Employee employee = (Employee) employeeCursor.get(0);
    employee.updateEmployee();
    session.update(employee);
    if ( ++count % 50 == 0 ) {
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

## Batch Processing Example:

Let us modify configuration file as to add **hibernate.jdbc.batch\_size** property:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
    </property>
    <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
    </property>

    <!-- Assume students is the database name -->
    <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
    </property>
    <property name="hibernate.connection.username">
      root
    </property>
    <property name="hibernate.connection.password">
      root123
    </property>
    <property name="hibernate.jdbc.batch_size">
      50
    </property>

    <!-- List of XML mapping files -->
    <mapping resource="Employee.hbm.xml"/>

  </session-factory>
</hibernate-configuration>
```

Consider the following POJO Employee class:

```
public class Employee {
  private int id;
  private String firstName;
  private String lastName;
  private int salary;

  public Employee() {}
  public Employee(String fname, String lname, int salary) {
    this.firstName = fname;
    this.lastName = lname;
    this.salary = salary;
  }
  public int getId() {
    return id;
  }
  public void setId( int id ) {
    this.id = id;
  }
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName( String first_name ) {
    this.firstName = first_name;
  }
  public String getLastName() {
    return lastName;
  }
  public void setLastName( String last_name ) {
    this.lastName = last_name;
  }
  public int getSalary() {
    return salary;
  }
}
```

```

    public void setSalary( int salary ) {
        this.salary = salary;
    }
}

```

Let us create the following EMPLOYEE table to store Employee objects:

```

create table EMPLOYEE (
    id INT NOT NULL auto_increment,
    first_name VARCHAR(20) default NULL,
    last_name VARCHAR(20) default NULL,
    salary INT default NULL,
    PRIMARY KEY (id)
);

```

Following will be mapping file to map Employee objects with EMPLOYEE table.

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Employee" table="EMPLOYEE">
        <meta attribute="class-description">
            This class contains the employee detail.
        </meta>
        <id name="id" type="int" column="id">
            <generator />
        </id>
        <property name="firstName" column="first_name" type="string"/>
        <property name="lastName" column="last_name" type="string"/>
        <property name="salary" column="salary" type="int"/>
    </class>
</hibernate-mapping>

```

Finally, we will create our application class with the main() method to run the application where we will use **flush()** and **clear()** methods available with Session object so that Hibernate keep writing these records into the database instead of caching them in the memory.

```

import java.util.*;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
    private static SessionFactory factory;
    public static void main(String[] args) {
        try{
            factory = new Configuration().configure().buildSessionFactory();
        }catch (Throwable ex) {
            System.err.println("Failed to create sessionFactory object." + ex);
            throw new ExceptionInInitializerError(ex);
        }
        ManageEmployee ME = new ManageEmployee();

        /* Add employee records in batches */
        ME.addEmployees();
    }
    /* Method to create employee records in batches */
    public void addEmployees() {
        Session session = factory.openSession();
        Transaction tx = null;
        Integer employeeID = null;
        try{

```

```

        tx = session.beginTransaction();
        for ( int i=0; i<100000; i++ ) {
            String fname = "First Name " + i;
            String lname = "Last Name " + i;
            Integer salary = i;
            Employee employee = new Employee(fname, lname, salary);
            session.save(employee);
            if( i % 50 == 0 ) {
                session.flush();
                session.clear();
            }
        }
        tx.commit();
    } catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    } finally {
        session.close();
    }
    return ;
}
}

```

## Compilation and Execution:

Here are the steps to compile and run the above mentioned application. Make sure you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

- Create hibernate.cfg.xml configuration file as explained above.
- Create Employee.hbm.xml mapping file as shown above.
- Create Employee.java source file as shown above and compile it.
- Create ManageEmployee.java source file as shown above and compile it.
- Execute ManageEmployee binary to run the program which will create 100000 records in EMPLOYEE table.